NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

PARTIAL-ENUMERATION FOR PLANAR NETWORK INTERDICTION PROBLEMS

by
Michael R. Boyle

March, 1998

Thesis Advisor: R. Kevin Wood
Second Reader: Gerald G. Brown

Approved for public release; distribution is unlimited

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1998	Master's Thesis

4. TITLE AND SUBTITLE
Partial-Enumeration for Planar Network Interdiction Problems

5. FUNDING NUMBERS

6. AUTHOR(S)
Boyle, Michael R.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

In the network interdiction problem, an interdictor destroys a set of arcs in a capacitated network through which an adversary will maximize flow. The interdictor's primary objective is to use his limited resources to minimize that maximum flow, but other objectives may be important. Therefore, we describe algorithms for enumerating near-optimal interdiction sets in planar networks so that these sets may be evaluated with respect to secondary criteria, e.g., safety of attacking forces, collateral damage, etc. Our algorithms are based on enumerating near-shortest paths or cycles in the dual of a planar network; they find a single optimal interdiction set in pseudo-polynomial time. We implement one algorithm applicable to *s-t* planar networks (*s* and *t* must lie on the perimeter of the network) and solve problems with up to 800 nodes and 1247 arcs. An example of computational results on that largest network is: The algorithm enumerates all 19 solutions that are within 50% of optimal in 0.15 seconds on a 300 mHz Pentium II PC. We also propose, but do not implement, a somewhat less efficient extension of this algorithm to solve problems on general planar networks.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Network Interdiction, Dijkstra's Algorithm	56
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFI- CATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

Approved for public release; distribution unlimited

PARTIAL-ENUMERATION FOR PLANAR NETWORK INTERDICTION PROBLEMS

Michael R. Boyle
Lieutenant, United States Navy
B.S., University of Illinois, 1988

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
March 1998

Author: MRB
Michael R. Boyle

Approved by: R. K. Wood
R. Kevin Wood, Thesis Advisor

Gerald G. Brown, Second Reader

Richard E. Rosenthal, Chairman
Department Of Operations Research

ABSTRACT

In the network interdiction problem, an interdictor destroys a set of arcs in a capacitated network through which an adversary will maximize flow. The interdictor's primary objective is to use his limited resources to minimize that maximum flow, but other objectives may be important. Therefore, we describe algorithms for enumerating near-optimal interdiction sets in planar networks so that these sets may be evaluated with respect to secondary criteria, e.g., safety of attacking forces, collateral damage, etc. Our algorithms are based on enumerating near-shortest paths or cycles in the dual of a planar network; they find a single optimal interdiction set in pseudo-polynomial time. We implement one algorithm applicable to *s-t* planar networks (*s* and *t* must lie on the perimeter of the network) and solve problems with up to 800 nodes and 1274 arcs. An example of computational results on that largest network is: The algorithm enumerates all 19 solutions that are within 50% of optimal in 0.15 seconds on a 300 mHz Pentium II PC. We also propose, but do not implement, a somewhat less efficient extension of this algorithm to solve problems on general planar networks.

TABLE OF CONTENTS

EXECUTIVE SUMMARY

This thesis develops a computer program to aid in effectively allocating limited resources to disrupt the maximum achievable flow of a single commodity in an adversary's planar, capacitated network. The network might be a transportation or pipeline network moving, respectively, war material or oil. The interdictor might be attacking the network with cruise missiles or aerial sorties.

Researchers have studied the network interdiction problem over many years, motivated originally by the desire to decrease an enemy's ability to wage war through reducing its ability to resupply. More recently, the U.S. government's attempt to curb the flow of illegal drugs into the country has driven advances in network interdiction techniques. These techniques focus on finding and implementing an optimal attack strategy on the network. However, an interdictor may be faced with a very complicated set of constraints, or at least a long list of desired qualities of an interdiction decision. For instance, the interdictor may be concerned about the safety of his attacking forces, or unnecessary collateral damage, etc. So, instead of trying to provide the single best solution with respect to a long list of criteria, we may prefer to provide the interdictor with a set of near-optimal solutions (with respect to some simple criterion such as maximum flow), and let the interdictor subsequently evaluate each of the candidate solutions in regard to the more complicated, perhaps even non-quantifiable criteria.

We address this void in the research by proposing two algorithms that enumerate near-optimal strategies. These algorithms build upon previous work in the field of network interdiction, and are divided into several sub-algorithms that incorporate typical "network techniques."

The first interdiction algorithm is applicable only to planar networks where both the sink and source lie on the outer perimeter of the network. Its first sub-algorithm constructs the dual network from the primal network. The dual is then expanded to deal with the consumption of varying, integral amounts of interdiction resource. The second solves the interdiction problem by finding a shortest path from the expanded dual source node to the closest expanded dual sink node. It does this by working backward from all the expanded dual sink nodes. In doing this computation, it also computes the shortest path distances from each expanded node to the closest expanded instance of the sink node. Using these distances, the third sub-algorithm enumerates near-optimal paths in the expanded dual network; these paths correspond to all of the near-optimal solutions of the network interdiction problem on the original network although some duplicates are created. These duplicates are easily culled.

We implement this algorithm and solve problems with up to 800 nodes and 1247 arcs. Representative computational results on this largest network are: The algorithm enumerates all 19 solutions that are within 50% of optimal in .15 seconds on a 300 mHz Pentium II PC.

The second interdiction algorithm we propose, but do not implement, is an extension of the first. It does require that the network be planar, but the source and sink nodes need not lie on the outer perimeter of the network. The algorithm should still be reasonably efficient: Computational complexity should increase, in practice, by a factor no larger than $2m$, where m is the minimum number of arcs among source-sink paths.

I. INTRODUCTION

This thesis develops a computer program to aid in effectively allocating limited resources to disrupt the maximum achievable flow of a single commodity in an adversary's planar, capacitated network.

A. THE NETWORK INTERDICTION PROBLEM

In the "network interdiction problem," an "interdictor" tries to minimize the maximum possible flow of a single commodity from a source node to a sink node in a capacitated network by efficiently using his limited assets to "interdict," i.e., break network arcs; the network user subsequently maximizes flow, subject to arc capacities, flow balance constraints and the fact that interdicted arcs are unusable, i.e., have zero capacity. Researchers have studied the network interdiction problem over many years, motivated originally by the desire to decrease an enemy's ability to wage war through reducing its ability to resupply (Wollmer 1964, Preston 1970). More recently, the U. S. government's attempt to curb the flow of illegal drugs into the country has driven advances in network interdiction techniques (Steinrauf 1991, Wood 1993).

The network interdictor is concerned with finding and implementing an optimal attack strategy on a network. Models in the literature have defined "optimal" to mean minimizing the maximum flow subject to the number of arcs destroyed being limited to a specified number, or subject to other simple constraints. However, an interdictor may be faced with a very complicated set of constraints, or at least a long list of desired qualities of an interdiction decision. Instead of trying to provide the single best solution with respect to a long list of criteria, we may prefer to provide the interdictor with a set of near-optimal solutions (with respect to some simple criterion such as maximum flow), and

let the interdictor subsequently evaluate each of the candidate solutions in regard to the more complicated criteria. Specifically, enumeration of near-optimal strategies, which we also call "partial enumeration," provides another degree of freedom for choosing a good attack strategy. For instance, the interdictor may not want to implement a strategy that isolates a specific node by destroying all adjacent arcs. In addition, the interdictor may be concerned about the safety of his attacking forces, or unnecessary collateral damage, etc. However, these strategies may be optimal in the simple, mathematical sense. Through partial enumeration, a strategy may be revealed that does not directly isolate the node, yet is within a few percent of being optimal with respect to a simple criterion – and the modest loss in "optimality" may not be critical to the interdictor's objectives. We will therefore explore techniques that will allow for the enumeration of near-optimal interdiction solutions.

B. BACKGROUND

Most work on the interdiction of flow networks is based on the max flow - min cut theorem, which states that the maximum flow in a capacitated network is equal to the capacity of the minimum capacity cutset (Ford and Fulkerson 1962). The network interdiction problem is then a problem of finding an attack strategy that minimizes the minimum capacity cut in the after-interdiction network.

Steinrauf (1991) and Wood (1993) both use mathematical programming techniques to solve this network interdiction problem. To illustrate their methods, let $G = (N,A)$ be a directed network with node set N and arc set A, where an arc is an ordered pair (i,j) with $i,j \in N$. Each arc (i,j) has an associated capacity u_{ij} and an interdiction resource expenditure r_{ij}. The decision variables are x_{ij}, the amount of flow the network user will

transport along the arc, and γ_{ij} a binary variable equal to 1 if the interdictor chooses to interdict the arc, and 0 otherwise. The network interdiction model can then be formulated as

$$\begin{aligned}
&\min_{\gamma\in\Gamma} \quad \max_{x} \quad x_{ts} \\
&\text{s.t.} \quad \sum_{j:(s,j)\in A} x_{sj} - \sum_{j:(j,s)\in A} x_{js} - x_{ts} = 0 \\
&\qquad \sum_{j:(i,j)\in A} x_{ij} - \sum_{j:(j,i)\in A} x_{ji} = 0 \quad \forall i \in N - s - t \qquad (1) \\
&\sum_{j:(t,j)\in A} x_{tj} - \sum_{j:(j,t)\in A} x_{jt} + x_{ts} = 0 \\
&\qquad 0 \le x_{ij} \le u_{ij}(1-\gamma_{ij}) \quad \forall (i,j) \in A
\end{aligned}$$

where $\Gamma = \{\gamma_{ij}: \gamma_{ij} \in \{0,1\}\ \forall (i,j) \in A,\ \sum_{(i,j)\in A} r_{ij}\gamma_{ij} \le R\}$. Note that by setting $\gamma_{ij} = 0\ \forall (i,j) \in A$, the inner maximization reduces to the standard maximum flow model. Our basic model is identical to this except that we assume arcs are undirected and the network must be planar. Wood also shows that by taking the dual of the inner maximization and linearizing, the above model can be converted into an integer program so that the problem can be solved using standard integer programming techniques.

The mathematical programming approach described above is very general. For instance, it can handle directed, undirected planar, and non-planar networks as well as multiple sources, sinks and resource types. However, the resulting models can be hard to solve (Cormican, 1995) and adding a partial enumeration capability could be difficult. Consequently, we base our techniques on simpler "network algorithms." These are algorithms, like shortest path algorithms, that work directly with network data structures rather than linear or integer programming algorithms that operate on an algebraic description of the problem.

Wollmer (1964) develops a network algorithm that finds the n arcs that, when removed from an undirected planar network, minimize the maximum flow. This algorithm makes use of a modified topological dual of the primal network (for a detailed discussion of the topological dual, see Ahuja, *et al.* 1993, pp. 262-265) where a new arc of zero length is added in parallel to each dual arc. Using a dynamic-programming method, the algorithm then calculates the shortest path through the modified dual network, traversing at most n arcs of zero length. The zero-length arcs correspond to those that should be interdicted in the primal network to minimize the maximum flow. Though Wollmer's algorithm solves the problem in polynomial time, it is limited to an *s-t* planar network, and it assumes that the dual is given and that every arc has an interdiction cost of one unit. (An "*s-t* planar network" has its source node s and sink node t on the periphery of the region enclosed by the network; s and t are positioned arbitrarily in a "general planar network.")

Phillips (1992) removes some of Wollmer's limitations and solves a variant of the network interdiction problem for undirected general planar networks with non-equal arc interdiction costs. She describes, but does not implement, three pseudopolynomial-time network algorithms that utilize dynamic programming. By introducing an error parameter $\varepsilon > 0$, she shows how a solution within $100\varepsilon\%$ of optimal can be returned in polynomial time.

C. PROPOSED ALGORITHMS

We propose two specialized network algorithms that enumerate near-optimal network interdiction solutions. The first algorithm applies to undirected *s-t* planar

networks, and the second to general undirected planar networks. We implement the first algorithm but not the second.

Our algorithm for undirected *s-t* planar networks provides a stand-alone tool that takes as its input the adversary's capacitated transshipment network along with arc capacities u_{ij}, interdiction costs r_{ij}, and a description of the embedding of that network in the plane. It then constructs the *s-t* dual network using an algorithm motivated by Bhaskar and Sahni (1988) for finding a rectangular dual of a planar triangulated graph. Our algorithm then solves for the optimal interdiction using an extension of Wollmer (1964). Finally, the algorithm enumerates near-optimal paths in an expanded dual network using a dynamic-programming technique similar to Byers and Waterman (1984) for finding near-optimal paths in a standard shortest path problem. Some duplicate solutions are created, but these are easily culled.

Our proposed methods are not as easy to generalize as the mathematical programming models of Steinrauf and Wood. However, we believe that the better speed at which such specialized algorithms as ours are likely to run, and the value of the enumerated near-optimal solutions, make them worth consideration.

D. THESIS OUTLINE

The remainder of this thesis is organized as follows. Chapter II gives essential definitions and notation. Chapter III develops the algorithm for *s-t* planar networks. Chapter IV shows how to extend the methodology to general planar networks but does not implement this extension. Chapter V investigates the performance of the algorithm of Chapter III on randomly generated networks of various sizes, and Chapter VI provides conclusions and recommendations for further research.

II. DEFINITIONS AND NOTATION

A. PRIMAL NETWORK

The network we wish to interdict is an undirected planar network with a source node s and sink node t. We will represent this network by $G = (N,A)$, and refer to it as the "primal network." N is the set of nodes, and A is the set of arcs (i,j) which are unordered, distinct pairs from N. We assume that the network is embedded in the plane, that all node positions are known, and that all arcs are non-crossing straight line segments with positive length. Each arc (i,j) has an associated integer capacity u_{ij} and an integer interdiction "cost" r_{ij}, i.e., a resource expenditure required to reduce flow capacity to 0 with probability 1. The network interdictor has an integer amount of resource R available for interdiction of this network. Note that any arc (i,j) can be denoted "uninterdictable" with $r_{ij} = \infty$.

The network user is trying to maximize the flow of a single commodity from s to the t in the primal network. (The inner maximization of Equation (1) with all $\gamma_{ij} = 0$.) This model is also valid for a capacitated undirected network if we make the following transformation to the undirected network: Replace every undirected arc (i,j) by two symmetric directed arcs (i,j) and (j,i), each with capacity u_{ij}, where u_{ij} is the capacity of the original undirected arc (e.g., Ahuja, *et al.* 1993, p. 168).

B. DUAL NETWORK

An undirected planar network $G = (N,A)$, with a given embedding in the plane, has associated with it a unique undirected planar network $G^* = (N^*,A^*)$. G^* is referred to as "the dual network." Each dual arc (i^*,j^*) uniquely intersects a primal arc, so that $|A^*| =$

$|A|$. The number of dual nodes $|N^*|$ is equal to the number of "faces" in the primal network. *A* "face" is a region defined by the arcs in a network, the unbounded region being the "outer face" (e.g., Lawler 1976, p. 33). For example, the sequence of nodes 1-2-6-5-1 in Figure 1 defines a face of *G*, and the sequence of nodes 1-2-3-4-8-7-6-5-1 defines the outer face of *G*.

The dual network for a general undirected (primal) network is constructed by placing a dual node in each face of the primal network, including the outer face, and then connecting dual nodes in adjacent faces with dual arcs. For our purposes, we will also assign "length" $u_{i^*j^*} = u_{ij}$ and resource $r_{i^*j^*} = r_{ij}$ to the dual arc (i^*,j^*) if it intersects primal arc (i,j). An example of this construction is shown in Figure 1. We will refer to this network as the "general dual network." Although the primal network's arcs can all be straight line segments, the dual arcs need not be.

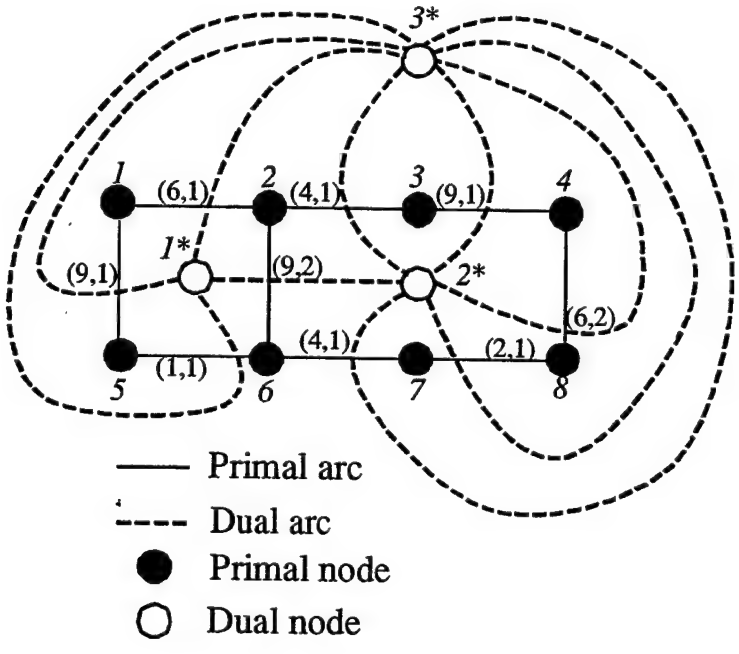


Figure 1. General Dual Network. General planar network and its dual. Primal arc labels are (u_{ij},r_{ij}).

When the primal network is *s*-*t* planar, we add an artificial arc from *s* to *t* to create an additional face. The dual network is then constructed by placing the dual source node *s** in this additional face, the dual sink node *t** in the outer face, and by constructing arcs as in the general case but with the omission of the arc (*s**,*t**). This construction is shown in Figure 2. We will refer to this network as the "*s**-*t** dual network."

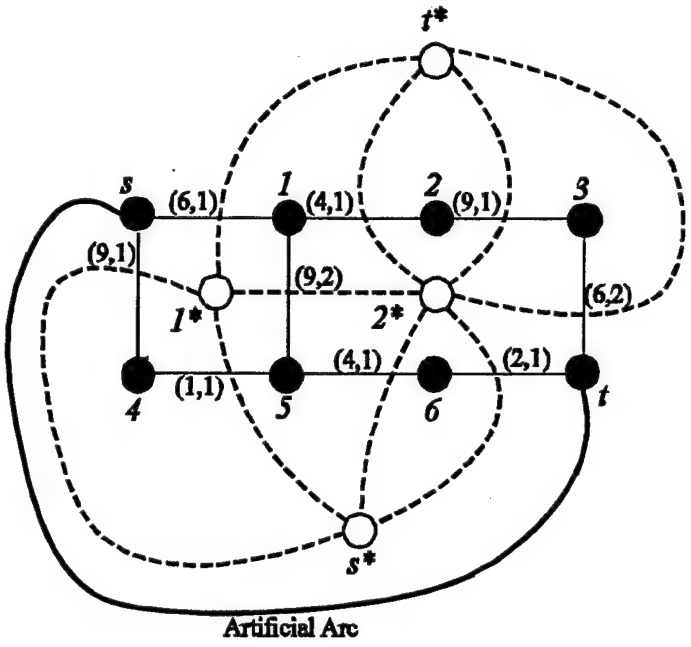


Figure 2. An *s-*t** Dual Network.** An *s*-*t* planar network and its dual. Primal arc labels are (u_{ij},r_{ij}).

The general dual network is of interest because of the one-to-one correspondence between *s*-*t* cutsets in the general primal network and simple cycles with odd parity in the dual network (e.g., Phillips 1992). A "cycle" is a path that begins and ends with the same node, and a "simple cycle" is a cycle which has no repeated nodes except the first. If we define *P* as the set of arcs in any simple path (no nodes may repeat) from *s* to *t* in the primal network, and *P** as the set of corresponding arcs in the dual network, "parity" is

defined as the number of arcs from the set P^* that are traversed in a dual cycle. By the Jordan curve theorem, a cycle separates nodes s and t if and only if it has odd parity (Phillips 1992). Therefore, identification of a minimum capacity s-t cutset in G becomes a problem of finding a shortest (in terms of the lengths $u_{i^*j^*}$) simple cycle in G^* with odd parity.

For s-t planar networks, a simple cycle with odd parity reduces to an s^*-t^* path in the s^*-t^* dual network (e.g., Ahuja, *et al.* 1993, p. 263). In terms of the s^*-t^* dual network, identification of the minimum capacity s-t cutset becomes the problem of finding a shortest path in G^* from s^* to t^* with respect to arc lengths $u_{i^*j^*}$.

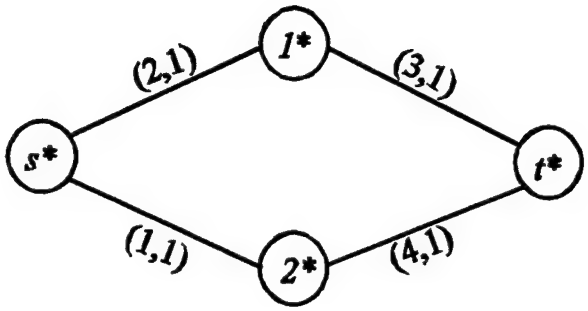
C. EXPANDED DUAL NETWORK

We now modify the dual network so that we can use a shortest path algorithm to solve the network interdiction problem. We call the modified dual network the "expanded dual network" and denote it as $G^E = (N^E, A^E)$. (The expanded dual network is an abstraction, and need not be created when actually solving the problem.) In addition, the modification varies slightly depending on whether the dual network is an s^*-t^* dual or a general dual. We will discuss the expanded s^*-t^* dual network first and then build upon that description for the expanded general dual network.

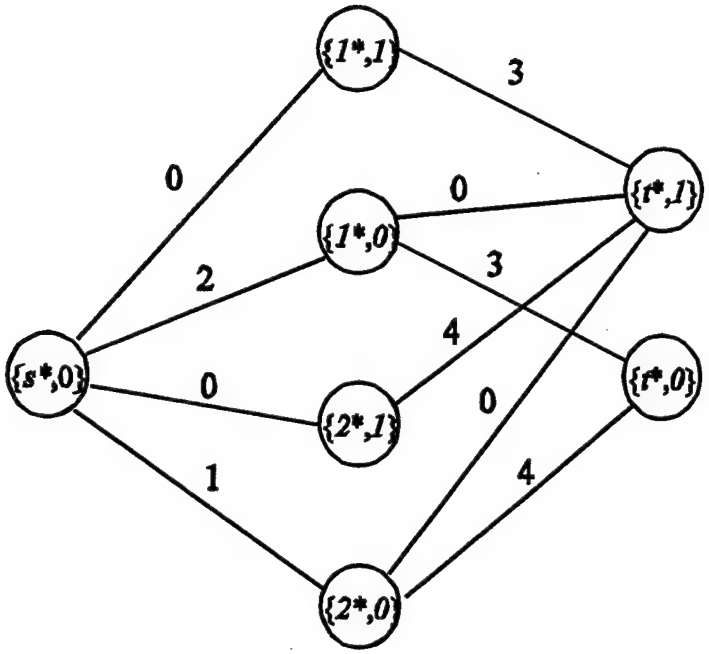
1. Expanded s^*-t^* Dual Network

Given that an interdictor has R units of resource available for interdiction, the expansion creates $R+1$ "expanded nodes" for each dual node. Each expanded node is identified by an ordered pair $\{i^*,r\}$ where i^* is the original dual node label and $r \in \{0,...,R\}$ is the level of resource expenditure. Starting from any expanded node $\{i^*,r\}$, there are two options given an incident arc (i^*,j^*) in the s^*-t^* dual network:

(a) Traverse the "expanded arc" $(\{i^*,r\},\{j^*,r\})$, whose length is $u_{i^*j^*}$, to the node $\{j^*,r\}$; this is equivalent to leaving the corresponding primal arc intact; or (b) if $r + r_{i^*j^*} \leq R$, traverse the expanded arc $(\{i^*,r\},\{j^*,r + r_{i^*j^*}\})$, whose length is 0, to the node $\{j^*,r + r_{i^*j^*}\}$; this is equivalent to interdicting the corresponding primal arc (Wollmer 1964). An example of a simple s^*-t^* dual network and the corresponding expanded s^*-t^* dual network with $R = 1$ is shown in Figure 3. The network interdiction problem in terms of the expanded s^*-t^* dual network reduces to finding the shortest of the shortest (distance) paths from $\{s^*,0\}$ to $\{t^*,r\}$ for $r \in \{0,...,R\}$.



(a)



(b)

Figure 3. Illustrating the expanded s^*-t^* dual network: (a) a simple s^*-t^* dual network with arc labels $(u_{i^*j^*}, r_{i^*j^*})$; (b) the associated expanded s^*-t^* dual network for $R = 1$ and arc labels $u_{i^*j^*}$.

2. Expanded General Dual Network

The expanded general dual network is a simple extension of the expanded s^*-t^* dual network. The extension creates two new nodes for every expanded s^*-t^* dual node, one for odd parity and one for even. If an arc $(i^*,j^*) \in P^*$, then its parity is 1; otherwise an arc's parity is 0. We then define the parity of a path to be the exclusive-OR of the parities of the arcs it contains. An expanded general dual node is identified by an ordered triple $\{i^*,r,p\}$ where p is the (binary) parity.

Starting from any expanded network node $\{i^*,r,p\}$, there are four general options given an incident arc (i^*,j^*) in the dual network:

(a) If $(i^*,j^*) \notin P^*$, traverse the expanded arc $(\{i^*,r,p\},\{j^*,r,p\})$, whose length is $u_{i^*j^*}$, to the node $\{j^*,r,p\}$; this is equivalent to leaving intact the corresponding primal arc $(i,j) \notin P$;

(b) If $(i^*,j^*) \notin P^*$ and $r + r_{i^*j^*} \leq R$, traverse the expanded arc $(\{i^*,r,p\},\{j^*,r + r_{i^*j^*},p\})$ whose length is 0 to the node $\{j^*,r + r_{i^*j^*},p\}$; this is equivalent to interdicting the corresponding primal arc $(i,j) \notin P$;

(c) If $(i^*,j^*) \in P^*$, and letting p' denote the parity opposite to p, traverse the expanded arc $(\{i^*,r,p\},\{j^*,r,p'\})$, whose length is $u_{i^*j^*}$, to the node $\{j^*,r,p'\}$; this is equivalent to leaving intact the corresponding primal arc $(i,j) \in P$;

(d) Or, if $(i^*,j^*) \in P^*$ and $r + r_{i^*j^*} \leq R$, traverse the expanded arc $(\{i^*,r,p\},\{j^*,r + r_{i^*j^*},p'\})$ whose length is 0 to the node $\{j^*,r + r_{i^*j^*},p'\}$; this is equivalent to interdicting the corresponding primal arc $(i,j) \in P$;

The network interdiction problem, for a general planar network, is therefore the problem of finding the shortest of the shortest (distance) paths in the expanded general dual network from $\{i^*,0,0\}$ to $\{i^*,r,1\}$ over all $i^* \in N^*$ and over all $r \in \{0,...,R\}$ (Phillips 1992).

III. INTERDICTION ALGORITHM FOR *s-t* PLANAR NETWORKS

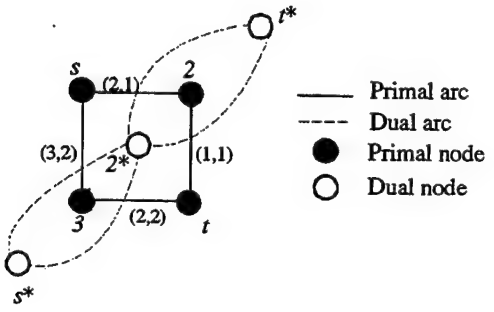
This chapter describes our network interdiction algorithm for to *s-t* planar networks. The algorithm consists of a main program that calls three procedures or "sub-algorithms." The first sub-algorithm constructs the dual network from the primal network. The second solves the interdiction problem by finding a shortest path from expanded node $\{s^*,0\}$ to the closest expanded node of the form $\{t^*,r\}$ by working backward from nodes $\{t^*,r\}$ for $r \in \{0,\ldots,R\}$. Simultaneously, it also computes the shortest path distances $f(j^*,r)$ from each expanded node $\{j^*,r\}$ to the closest expanded instance of t^*. Using the values $f(j^*,r)$, the third sub-algorithm enumerates near-optimal paths in the expanded dual network; these paths correspond to all of the near-optimal solutions of the network interdiction problem on the original network; some duplicate interdiction solutions may be generated, but these are easily culled. We first describe the data structures used by the sub-algorithms and state assumptions regarding input data.

A. DATA STRUCTURES

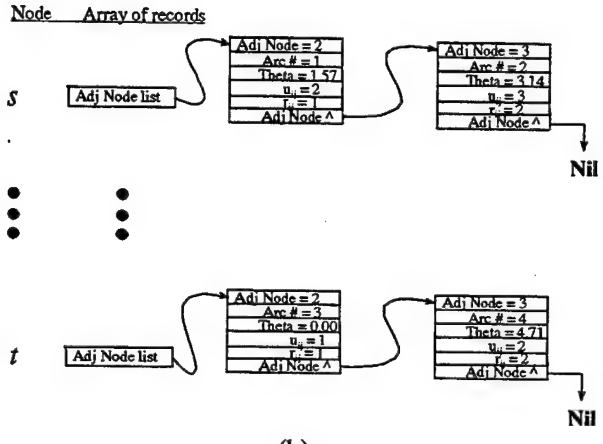
We assume that the primal network is embedded in the plane so that no arcs cross. We assume that all node positions are known, and that all arcs are straight line segments with positive length. The input file contains the number of nodes, the Cartesian coordinates of each node, and for each arc (i,j) the associated values i, j, u_{ij}, and r_{ij}.

We utilize three main data structures to implement our interdiction algorithm. The first of these is a network data structure used to store the primal network; this is the "primal data structure." This data structure uses an array of linked lists, one list for each primal node. The linked list for each node i contains a record for each adjacent node j sorted by increasing θ_{ij}; θ_{ij} is defined as the angle in radians measured clockwise from the

positive y-axis to the arc (i,j) with the Cartesian coordinate system centered at node i. Using Figure 1 (in Chapter II), two examples are: $\theta_{12} = \pi/2$ radians, and $\theta_{26} = \pi$ radians. Each record in the linked list for node i contains a field for the adjacent node number j, the associated arc number, θ_{ij}, u_{ij}, r_{ij}, and a pointer to the next adjacent node or a null pointer if there are no more adjacent nodes. For an example of a primal data structure, see Figure 4. The primal data structure is filled with data from an input file.



(a)



(b)

Figure 4 The primal data structure. The first node is s and the last is t. (a) an s-t (primal) network with arc labels (u_{ij},r_{ij}) and the associated dual; (b) the associated primal data structure for nodes s and t.

The second data structure exploits the fact that each arc in the primal network has associated with it a unique arc in the dual network; this structure is the "arc data structure." Its purpose is to allow quick output of a solution in terms of the primal network despite the fact that the solution was obtained by manipulation of the dual network. The arc data structure consists of an array of records with one record for each primal arc (i,j), each record containing a field for i, j, u_{ij}, r_{ij}, and the dual nodes i^*, j^*, which correspond to the crossing dual arc (i^*,j^*). The first four fields are filled with primal network data from the input file, and the last two are filled as the dual network is created.

The third data structure is identical to the first except that it is filled by reading the dual network from the arc data structure detailed above once the dual has been created by our dual-finding algorithm; this structure is the "dual data structure." Although we do not create an explicit representation of the expanded dual network, we do need to store node information for that network, namely, the values $f(j^*,r)$. This is done in a two-dimensional array with R $|N|$ elements.

B. DUAL-FINDING ALGORITHM

The first sub-algorithm in our network interdiction algorithm constructs the dual network. Discussion of the topological dual of a planar network is ubiquitous in the network literature (e.g., Ahuja, *et al.*, 1993, pp. 262-265, Lawler, 1976, pp. 32-36). However, we are unable to locate in the literature an algorithm that constructs the dual of a planar network. Bhasker and Sahni (1988) do describe an algorithm to construct the dual of a planar triangulated network. Unfortunately, the triangulated assumption is too restrictive for our work as it requires: Every face is a triangle, all internal nodes have

degree ≥ 4, and all cycles that are not faces have length ≥ 4. However, their algorithm does provide a starting point, and given this starting point and O'Rourke's (1994) brief discussion, we are able to derive an algorithm to construct the dual of an *s-t* planar network.

Given an *s-t* planar network (as in Figure 5(a)), it is easy to construct the dual (Figure 5(b)) using the techniques discussed in Chapter II.

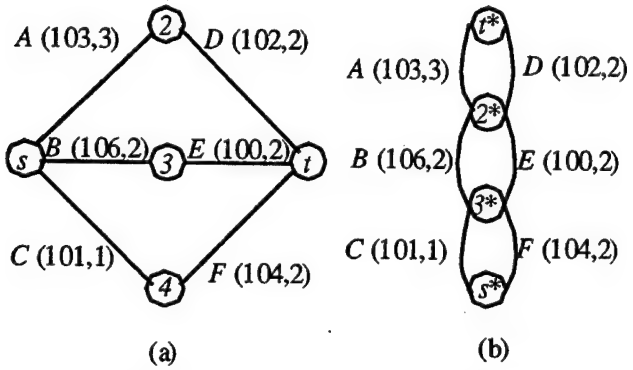


Figure 5. An *s-t* primal network (a) and its dual (b) with arc identifiers "*I*," and arc data $I(u_{ij},r_{ij})$.

The key element in constructing the dual is to recognize a face in the primal network. As each inner face corresponds to a cycle containing no subcycles, the algorithm must identify these special cycles. Our algorithm operates as follows.

Since each arc lies on the perimeter of two faces, a new directed network G' is constructed from G by replacing each arc (i,j) with two directed arcs in anti-parallel, (i,j) and (j,i). In addition, we create two artificial directed arcs, (s,t) and (t,s). These arcs represent the artificial arc (s,t) that wraps around G and is used in creating the s^*-t^* dual. See Figure 2 in Chapter II. Starting at s, a directed path is then traversed in G' under the constraint that doubling back is disallowed, i.e., the arc (j,i) can not be added to the path

immediately after traversing arc (i,j). When an arc (i,j) is traversed to reach a new node j, the next arc traversed (j,k) must form the sharpest possible angle, in a clockwise direction, with (i,j). (At a junction, always take the sharpest turn to the right that is available.) When some node i is repeated in the path, a special cycle, i.e., a face, has been identified. All arcs in that cycle are deleted, and the search is continued from i. If i has no incident arcs remaining, a node with incident arcs is identified and the search is restarted from there. Once all arcs in the network have been deleted, all faces have been identified.

To implement the algorithm, we utilize the primal data structure which stores each arc twice. We do not actually wait until a cycle has been identified to delete arcs, but rather, we delete them as they are traversed. (The current path is stored in a stack, so nothing is lost by immediately deleting arcs.) The algorithm continues to look for faces until no arcs remain. For example, the dual of the primal in Figure 5(a) would be found as follows. Beginning at s ($s = 1$), the algorithm first traverses to node 2, then to node t, then to node 3 and then back to s. As this completes a path from s to s, a face is found which encloses dual node 2^*. The algorithm then looks for the next face. Since s has untraversed incident arcs, and since it is where the cycle was identified, the search restarts from there, traverses to node 3, then to node t, then to node 4, and back to node s locating the second face, i.e. dual node 3^*. The algorithm continues to return to node s until all incident arcs have been deleted from it. The algorithm then increments to the next node $j = 2$ until all incident arcs have been deleted from it, then to $j = 3$, etc. When no primal arcs remain, all the dual network nodes have been located.

The dual construction is completed by creating the dual arcs in a process that exploits the one-to-one correspondence between primal and dual arcs. We utilize the arc data

structure which stores i, j, u_{ij}, r_{ij}, i^*, and j^* for each of these arc pairs. The first four items we assume are given, the last two we define as the dual nodes are found. For example, when we have identified a face (which corresponds to dual node k^*) with primal arcs (i_0, i_1), (i_1, i_2), ..., (i_{t-1}, i_0), we set $i^* = k^*$, or $j^* = k^*$ if i^* is already specified, for each arc in the path identifying the face.

1. Assumptions

The algorithm closely follows the discussion above, but there are a few details that should be expanded. In order to construct the dual network, it is vital for us to know the location of both s and t in G. Without this information we would be unable to construct the artificial arc which wraps around G and connects s to t. By definition of an s-t planar network, both s and t lie on the outer perimeter of the network. Since s and t lie on the outer face of G, there exist angles θ_{st} and θ_{ts} respectively, such that a line segment with origin at s (at t) and extending at an angle of θ_{st} (θ_{ts}) does not intersect G. Clearly, these line segments can be joined into an artificial arc (s,t) which bends around G without intersecting it. Our algorithm assumes that θ_{st} and θ_{ts}, as described, are given as inputs.

The pseudo-code for the algorithm follows. It uses $\phi_{ijk} \equiv \theta_{jk} - \theta_{ij}$. (For example, ϕ_{126} is $\pi/2$ radians in Figure 1 in Chapter II.)

2. Pseudo-Code for Dual-Finding Algorithm

```
Procedure: FINDDUAL(G)
Input:   An undirected planar network G = (N,A) in adjacency-list form.
         (Arcs incident to each node are sorted by increasing θ_ij.)
         Arc[A] /* an array of records containing i, j, u_ij, r_ij */
Output: G* = (N*,A*), the dual network associated with G
         in adjacency list form;
{
        Add artificial arcs (s,t) and (t,s) to G;
        Mark all i ∈ N as "offcycle";
        DualNodeCounter ← 0;
        For m = 1 to N {
                i ← m;
                Push i on top of stack S;
                Mark i as "oncycle";
                While i still has an incident arc {
                        Select an arc (i,j) which maximizes φ s.t. φ ≤ π radians;
                        If no such arc exists, choose the first arc in adjacency list;
                        Delete arc (i,j) from the adjacency list;
                        If j "oncycle" /* face found */ {
                                DualNodeCounter= DualNodeCounter+1
                                Pop all nodes from S until top of S = j;
                                Mark all nodes popped as "offcycle";
                                For each arc (i,j) that was "oncycle" {
                                        Arc[(i,j)].i* ← DualNodeCounter;
                                }
                                i ← j;
                        }
                        else {
                                Push j on top of stack S;
                                Mark j as "oncycle";
                                i ← j;
                        }
                }/* j has no more adjacent nodes */
                Pop i from S;
                Mark i "offcycle";
        }
}
```

3. Complexity Analysis

Each arc (i,j) is "selected" once, and this selection process, assuming bounded degree, takes O(|1|) time. Therefore, there is $O(|A|)$ total work associated with selecting arcs. Once an arc is selected, there is only O(|1|) additional work, so the complexity of this algorithm is $O(|A|)$. Without the assumption of bounded degree, the complexity is $O(|N||A|)$

C. DIJKSTRA'S SHORTEST PATH ALGORITHM

The second sub-algorithm now computes the optimal solution to the interdiction problem. This solution corresponds to the shortest of the paths in the expanded dual network from the expanded node $\{s^*,0\}$ to nodes of the form $\{t^*,r\}$. We accomplish this by working backward in an implicit representation of the expanded dual network using a binary-heap implementation of Dijkstra's shortest path algorithm (e.g., Ahuja, *et al.* 1993, pp. 115-116). The shortest path distance from any instance t^* to itself is 0, so the "reverse shortest path distance" $f(t^*,r)$ is initialized to 0 for all r, and the heap is initialized with all of these expanded nodes as starting nodes. In computing the optimal interdiction solution, the algorithm also computes the shortest path length from every expanded dual node $\{j^*,r\}$ to the closest expanded instance of t^*. We denote these distance as $f(j^*,r)$, and the shortest path length (optimal solution value) as $f^* = f(s^*,0)$.

Importantly, the shortest path distances $f(j^*,r)$ are exactly what is needed for the partial-enumeration sub-algorithm.

The algorithm proceeds in an obvious manner except that the existence of expanded arcs is checked on the fly: Suppose that the algorithm has just removed

expanded node $\{j^*,r\}$ from the heap and must scan arcs incident to this node. For each arc (i^*,j^*) in the regular dual network, the expanded dual arc $(\{i^*,r\},\{j^*,r\})$ with length $u_{i^*j^*}$ will typically exist and may be scanned since it corresponds to not interdicting some primal arc (i,j) and not consuming any resource. (An exception occurs when $i^* = s^*$ and $r > 0$.) On the other hand, the expanded arc $(\{i^*,r - r_{i^*j^*}\},\{j^*,r\})$ cannot exist and will not be scanned if $r - r_{i^*j^*} < 0$, or $r - r_{i^*j^*} > 0$ and $i^* = s^*$.

The complexity of a binary-heap implementation of Dijkstra's algorithm is $O(|A|\log|N|)$ (e.g., Ahuja, *et al.* (1993), p. 116). Our modification essentially runs on the expanded dual network which has $R|N|$ nodes and $2R|A|$ arcs. Therefore the complexity of our algorithm is pseudo-polynomial. We do not include pseudo-code for the algorithm since it is straightforward.

D. PARTIAL-ENUMERATION ALGORITHM

The third sub-algorithm is based on Byers and Waterman (1984). Their algorithm enumerates all *s-t* paths within a prescribed distance of the optimal path length from source to sink in a directed acyclic network. They modify a (total) path-enumeration algorithm (which is just a modified depth-first search) to locate these paths, and a push-down stack to store all current valid paths. (The algorithm is related to an "A* search," e.g., Hart, *et al.*, 1968.) We modify this algorithm to work on an implicit representation of the expanded dual network.

The partial-enumeration sub-algorithm finds near-optimal solutions by traversing all paths from $\{s^*,0\}$ to $\{t^*,r\}$,for $r \in \{0,\ldots,R\}$ that are within $100\varepsilon\%$ of optimal. Let $S_{\{i^*,r\}}$ denote any simple path from $\{s^*,0\}$ to $\{i^*,r\}$, define $d(S_{\{i^*,r\}})$ as its length, and define

$R(S_{\{i^*,r\}})$ as the total resource expended to traverse $S_{\{i^*,r\}}$. R units of resource are assumed available for interdiction.

Given a partial path $S_{\{i^*,r\}}$, the algorithm determines if

$$d(S_{\{i^*,r\}}) + u_{i^*j^*} + f(j^*,r) \leq (1+\varepsilon)f^*. \qquad (2)$$

If it is, the algorithm feasibly extends the path $S_{\{i^*,r\}}$ by adding the uninterdicted arc (i^*,j^*) and continues the search from $\{j^*,r\}$ with partial path $S_{\{j^*,r\}}$. This is equivalent to traversing the arc $(\{i^*,r\},\{j^*,r\})$ in the expanded dual network. If it is not, the algorithm determines if

$$R(S_{\{i^*,r\}}) + r_{i^*j^*} \leq R \text{ and,} \qquad (3)$$

$$d(S_{\{i^*,r\}}) + f(j^*, r + r_{i^*j^*}) \leq (1+\varepsilon)f^*. \qquad (4)$$

If those conditions are met, the algorithm extends $S_{\{i^*,r\}}$ by adding the interdicted arc and continues the search from $\{j^*, r + r_{i^*j^*}\}$ with partial path $S_{\{j^*,r+r_{i^*j^*}\}}$. This is equivalent to traversing the arc $(\{i^*,r\},\{j^*,r+ r_{i^*j^*}\})$ in the expanded dual network. The algorithm continues this recursion until all near-optimal paths have been enumerated. This method can produce the same interdiction set more than once because it enumerates s-t cuts in G that may not be minimum. Consequently, duplicate, non-minimum solutions must be culled.

As an example, consider the network illustrated in Figure 5, when we set $R = 3$ and $\varepsilon = .06$. We obtain an optimal solution of value 100, two near-optimal solutions and three duplicates; all six "solutions" are listed in Table 1.

Arcs Crossing Optimal and Near-Optimal Cutsets	Residual *s-t* Maximum Flow
$(1,4)'$ $(3,5)$ $(2,5)'$	100
$(1,4)'$ $(1,3)'$ $(2,5)$	102
$(1,4)'$ $(3,5)'$ $(2,5)$	102
$(1,4)'$ $(1,3)'$ (1,2) duplicate	103
$(1,4)'$ $(3,5)'$ (1,2) duplicate	103
$(1,4)'$ (1,3) $(2,5)'$ duplicate	106

Table 1. Solution for primal network shown in Figure 5 with s = 1, t = 5, R = 2 and ε = .06. The notation (i,j) indicates that the arc is not interdicted; and $(i,j)'$ indicates that it is.

We demonstrate how the last solution was completed assuming that $S_{(i^*,r)} = \{\{s^*,0\}, \{3^*,1\}\}$, so that $d(S_{(i^*,r)}) = 0$ and $R(S_{(i^*,r)}) = 1$:

(a) The algorithm will traverse to $\{2^*,1\}$ via the uninterdicted arc B if formula (2) is true. As $0 + 106 + 0 \leq (1 + .06)100$ the algorithm does add the node $\{2^*,1\}$ to the path.

(b) The algorithm will traverse to $\{t^*,1\}$ via the uninterdicted arc A if formula (2) is true. As $106 + 103 + 0 \geq (1 + .06)100$ the algorithm cannot add the node $\{t^*,1\}$ to the path.

(c) The algorithm will traverse to $\{t^*,4\}$ via the interdicted arc A if (3) and (4) are true. As $1 + 3 \geq 3$ the algorithm cannot add the node $\{t^*,4\}$ to the path.

(d) The algorithm will traverse to $\{t^*,1\}$ via the uninterdicted arc D if formula (2) is true. As $106 + 102 + 0 \geq (1 + .06)100$ the algorithm cannot add the node $\{t^*,1\}$ to the path.

(e) The algorithm will traverse to $\{t^*,3\}$ via the interdicted arc D if both formulas (3) and (4) are true. As $1 + 2 \leq 3$ and $106 + 0 + 0 \leq (1 + .06)100$ the algorithm does add the

node $\{t^*,3\}$ to the path. Because this is a sink node, the algorithm outputs the path in terms of the primal network as shown in the last row of Table 1. The path is shown in terms of the dual in Figure 6.

(f) Because every arc incident to $\{2^*,1\}$ has been investigated, the algorithm now backtracks to node $\{3^*,1\}$ and continues investigating uninvestigated arcs.

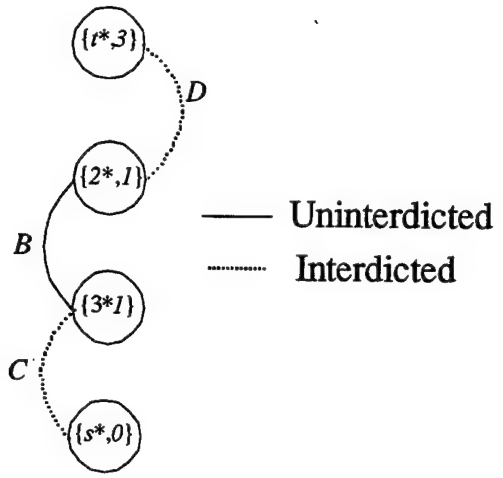


Figure 6. Illustrating a solution as a dual path with arc labels *I*.

Let S be a stack that stores pairs $<j^*,b>$ where j^* is a node on the path being enumerated and b indicates the status of the arc, say (i^*,j^*), that reaches j^* on the current path from s^*: $b = 1$ implies that the corresponding primal arc (i,j) is interdicted; otherwise $b = 0$. The pseudo-code for the algorithm follows.

1. Pseudo-Code

```
Procedure: ENUMERATE(G*, f, f*, ε)
Input: Dual network G *= (N*,A*) in adjacency list form, f(i*,r) ∀ i*∈N, f* and ε;
Output: All solutions to the network interdiction problem with capacity (1 + ε)f*;
{
    Mark all i*∈ N* − s* as, "off path"; Mark s* as "on path";
    Mark all (i*,j*) ∈ A* as "untraversed";
    S ← ∅;
    Push <s*,−1> on top of stack S; /* "−1" indicates undefined */
    /* (S, <t*,b>) will be an encoding of the dual path from s* to t* along with the status of the
    arcs on that path */
    R(S{s*,r}) ← 0;
    While S ≠ ∅ {
        <i*,b> ← top of S;
        While not all arcs out of i* are "traversed" {
            Select an "untraversed" arc (i*,j*);
            If j* is "off path" {
                If d (S{i*,r}) + u_i*j* + f(j*,r) ≤ (1 + ε)f* {
                    /* arc is not interdicted */;
                        If j* = t* { Print(S, <t*,0>) };
                        else { Push <j*,0> on S;
                            Mark j* as "on path";
                            i* ← j*;
                        }
                }
                else {
                    If R(S{i*,r}) + r_i*j* ≤ R {
                        If d (S{i*,r}) + f(j*,r + r_i*j*) ≤ (1 + ε)f* {
                            /* arc is interdicted */
                            If j* = t* { Print(S, <t*,1>) };
                            else { Push <j*,1> on S;
                                Mark j* as "on path";
                                i* ← j*;
                            }
                        }
                    }
                }
                If arc has been interdicted { Mark (i*,j*) as "traversed"; }
            }
        }
        /* all arcs out of i* have been "traversed" */
        Mark all arcs out of i* as "untraversed";
        Mark i* as "off path";
        Pop <i*,b> from S;
    }
}
```

2. Complexity Analysis

It is easy to construct a planar network so that its dual has $O(2^{|N|})$ paths between two nodes s^* and t^*. If most of these paths correspond to candidates for near-optimal solutions to the network interdiction problem in the primal network (and this can be arranged), then any algorithm to enumerate these paths must be exponential. As each path in the expanded dual network might take $O(R|A|)$ work to find, the complexity of the partial-enumeration algorithm could be $O(R|A|2^{|N|})$. In practice, the run time is strongly dependent on ε and the number of paths enumerated. If ε is not too large, and only a few near-optimal paths exist, the algorithm is quite efficient. In practice ε could be incrementally increased from a very small value until the number of near-optimal paths becomes suitably large, but still manageable.

We perform culling of duplicate solutions off-line using a spreadsheet sort and do not include it complexity. Clearly, this procedure could be performed efficiently using a has function and table as candidate solutions are generated.

In Chapter V we will provide computational results for the interdiction algorithm described in this chapter. Before we do that, however, we describe an algorithm to solve the interdiction problem on non *s-t* planar networks.

IV. ALGORITHM FOR GENERAL PLANAR NETWORKS

This chapter describes, but does not implement, our network interdiction algorithm applicable to general *s-t* planar networks (i.e., planar networks where *s* and *t* do not necessarily lie on the perimeter of the network). We propose four sub-algorithms: The first conducts a breadth-first search to identify a simple path *P* between *s* and *t* in the primal network. The second constructs the (general) dual network from the primal network. The third sub-algorithm, a variant of Dijkstra's algorithm, operates on an expanded dual that incorporates parity, i.e., the number of times that a path crosses *P*. To solve the interdiction problem, this sub-algorithm must be called once for each dual arc (i^*,j^*) in a special subset of dual arcs. A set of restricted optimal values $f^*_{(i^*,j^*)}$ is computed in this way so that the global optimum is $f^* = \min_{(i^*,j^*)} f^*_{(i^*,j^*)}$. The fourth sub-algorithm performs enumeration much as in the previous chapter, but must be called for each special arc (i^*,j^*) with $f^*_{(i^*,j^*)} \leq (1+\varepsilon)f^*$. Also, before each enumeration call, the third sub-algorithm must be rerun to compute $f_{(i^*,j^*)}(j^*,r,p)$ that denotes the length of the shortest path from expanded node $\{j^*,r,p\}$ (p is parity) to the closest node $\{i^*,r,1\}$ for $r \in \{0,\ldots,R\}$.

A. BREADTH-FIRST SEARCH

The identification of a minimum capacity *s-t* cutset in a non-*s-t*-planar network is equivalent to finding the shortest simple cycle in the dual network with odd parity. Here, parity counts the number of times the cycle crosses a specified, simple *s-t* path. (See Chapter II.) Our interdiction algorithm is similarly dependent on finding cycles of odd parity. Any simple path between *s* and *t* will do for counting parity, but for efficiency's

sake we prefer a path with a minimum number of arcs. We identify such a path in $O(|A|)$ time using a standard breadth-first search (e.g., Cormen *et al.*, 1992, pp. 469-475).

B. DUAL-FINDING ALGORITHM

Constructing the dual of a general planar network is identical to constructing the s^*-t^* dual as described in Chapter III except: No artificial arc (s,t) is added to G and thus, the directed arcs (s,t) and (t,s) are not added to G' (the directed network used in constructing the dual G^*).

C. A MODIFIED DIJKSTRA'S SHORTEST PATH-ALGORITHM

The third sub-algorithm, a variant of Dijkstra's shortest path method, will be called multiple times to solve the interdiction problem and to help enumerate near-optimal solutions. The optimal solution to the interdiction problem corresponds to the shortest possible path from some expanded node $\{i^*,0,0\}$ to a node of the form $\{i^*,r,1\}$ for $r \in \{0,\ldots,R\}$. We could call the shortest path algorithm for each $i^* \in N^*$ to solve the interdiction problem optimally, but the search can be limited further, as described next.

An optimal interdiction solution corresponds to a cycle in the dual that crosses path P an odd number of times. We may view the optimal cycle as starting at some dual node i^*, immediately traversing dual arc (i^*,j^*) that crosses arc $(i,j) \in P$, and then eventually returning to i^*. Thus, we could solve the interdiction problem by examining all shortest cycles that immediately cross from one side of P to the other:

For each dual arc (i^*,j^*) crossing P:

> Find a shortest path from $\{i^*,0,0\}$ that traverses an expanded instance of (i^*,j^*) to a node of the form $\{j^*,r',1\}$ and eventually returns to a node of the form $\{i^*,r,1\}$ for $r \in \{0,\ldots,R\}$.

The global optimum corresponds to the shortest shortest path found. Let A^{**} denote the special set of arcs from which we initiate the modified shortest path algorithm, as above. We would normally expect that $|A^{**}| << |N^*|$, so significant efficiency should be gained by limiting the shortest path searches as described.

The shortest path algorithm must also be modified to consider only certain arcs leaving the source node which is $\{i^*,0,0\}$in this case. This modification is trivial: From$\{i^*,0,0\}$, scan only "allowable" arcs whose head nodes are of the form $\{j^*,r,1\}$. Actually, as in the *s-t* planar case, the shortest path algorithm will be working backwards in the expanded network to reach $\{i^*,0,0\}$. Therefore, the shortest path search is started simultaneously from nodes of the form $\{i^*,r,1\}$ and initially scans arcs entering from nodes $\{j^*,r',0\}$ only.

Our choice for a shortest path algorithm is the modified version of Dijkstra's method that operates on the (implicit) expanded dual network, as in Chapter III, but is modified further. The algorithm must be adjusted to compute parity properly and to implement the special scanning rules described above. The scanning rules are straightforward to implement; handling parity needs more explanation.

Like our original shortest path algorithm, this one works backwards, in this case from nodes of the form $\{i^*,r,1\}$for $r \in \{0,\ldots,R\}$ to node$\{i^*,0,0\}$. The algorithm minimizes

$f_{(i^*,j^*)}(h^*,r,p)$ which is defined as the shortest path distance from expanded node $\{h^*,r,p\}$ to the closest node of the form $\{i^*,r',1\}$. (We use h^* and k^* as a generic nodes here to avoid confusion with the nodes i^* and j^* used in the previous paragraph.) The algorithm handles parity as follows: Suppose the algorithm is scanning the two expanded arcs, which correspond to a dual arc (h^*,k^*), entering the expanded node $\{k^*,r,p\}$. If (h^*,k^*) does not cross the path P, the algorithm determines if $f_{(i^*,j^*)}(h^*,r,p)$ and $f_{(i^*,j^*)}(h^*,r-r_{h^*k^*},p)$ are to be updated. (We have scanned the implicit expanded arcs $(\{h^*,r,p\},\{k^*,r,p\})$ and $(\{h^*, r-r_{h^*k^*},p\}, \{k^*,r,p\})$.) If (h^*,k^*) crosses P, the algorithm determines if $f_{(i^*,j^*)}(h^*,r,1-p)$ and $f_{(i^*,j^*)}(h^*,r-r_{h^*k^*},1-p)$ are to be updated. (We have scanned the implicit expanded arcs $(\{h^*,r,1-p\}, \{k^*,r,p\})$ and $(\{h^*, r-r_{h^*k^*},1-p\}, \{k^*,r,p\})$.)

We denote the optimal solution value determined for arc $(i^*,j^*) \in A^{**}$, as above, by $f^*_{(i^*,j^*)}$ and let $f^* = \min_{(i^*,j^*)} f^*_{(i^*,j^*)}$. So, f^* is the optimal solution value to the network interdiction problem on G. We will call the shortest path algorithm again, multiple times, from within the enumeration algorithm described next.

D. PARTIAL-ENUMERATION ALGORITHM

The fourth sub-algorithm differs from the enumeration algorithm detailed in Chapter III in that it must be called for each arc $(i^*,j^*)\in A^{**}$. For each such node, it begins by calling the third sub-algorithm to compute the values $f_{(i^*,j^*)}(h^*,r,p)$. Once these values are obtained, the algorithm searches the expanded network – arcs are generated on the fly, of course – looking for paths from $\{i^*,0,0\}$ to $\{i^*,r,1\}$ that are within $100\varepsilon\%$ of optimality. This procedure is identical to the partial-enumeration algorithm in Chapter III except that it must account for node parity, just as the shortest path algorithm does.

We conjecture that the work performed by both the shortest path and partial-enumeration sub-algorithms might be reduced if those procedures could be started from a set of arcs that is a proper subset of A^{**}. Suppose we have just identified an optimal cycle that starts at $(i_1^*, j_1^*) \in A^{**}$ and then recrosses P twice using $(i_2^*, j_2^*) \in A^{**}$ and $(i_3^*, j_3^*) \in A^{**}$, say. (It must cross P an odd number of times.) Then, we need not begin a search for an optimal cycle starting with (i_2^*, j_2^*) or (i_3^*, j_3^*) since the cycle we just described would just be found again. Thus, the number of calls of Dijkstra's sub-algorithm for computing f^* could definitely be reduced. In this solution, it may also be possible to reduce the number of calls to the enumeration sub-algorithm, but this is less certain and needs further investigation.

Culling of candidate near-optimal solutions is performed the same way it is for *s-t* planar networks.

V. COMPUTATIONAL RESULTS FOR THE INTERDICTION ALGORITHM

In this chapter we demonstrate the operation of our interdiction algorithm for *s-t* planar networks on four randomly generated problems. The purpose is to demonstrate accuracy and efficiency, and potential usefulness to an analyst in choosing a network interdiction strategy. We run the algorithm on a small network to illustrate what an enumeration output looks like and to allow the reader to verify correctness. We run the algorithm on three large networks of varying size to demonstrate computational efficiency, and thus the potential for solving large, real-world networks.

A. ALGORITHM IMPLEMENTATION AND NETWORK GENERATION

Our algorithm is coded and compiled using the Borland C++ Builder 3 and run on a 300 mHz Pentium II PC. We also devise a C++ program to randomly generate a planar grid network. For an input value n, the program generates an n by $2n$ grid network. All arcs between adjacent nodes on the perimeter of the network are added; this defines the outer face. In addition, all other nodes that are adjacent horizontally are connected by arcs. The program then completes the network by randomly constructing about 60% of the vertical arcs in the interior. The program randomly assigns r_{ij} and u_{ij}, uniformly in ranges specified by the user. It then writes this data to an output file that the interdiction algorithm can read. An example of a randomly generated network for $n = 3$, $u_{ij} \in \{95,\ldots,104\}$, and $r_{ij} \in \{1,2\}$ is shown in Figure 7; this is also the network we will refer to as the "small network."

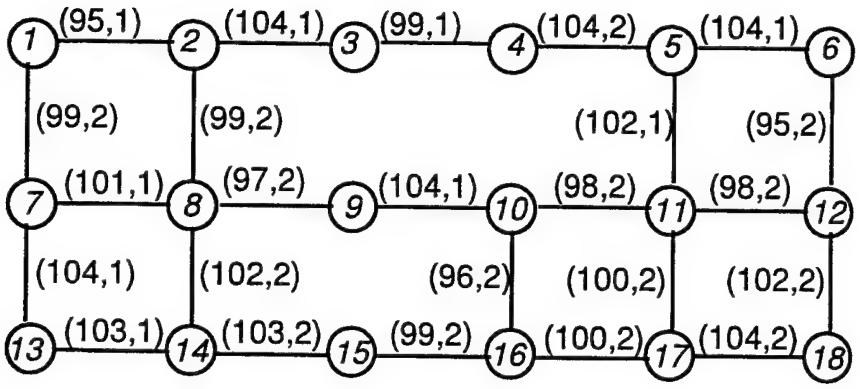


Figure 7. Small Network. The primal network generated for $n = 3$, $s = 1$, $t = 18$, and arc data (u_{ij},r_{ij}).

B. COMPUTATIONAL RESULTS FOR THE SMALL NETWORK

Utilizing the primal network depicted in Figure 7, we run our interdiction algorithm for $R = 2$ and $\varepsilon = .05$, and $R = 2$ and $\varepsilon = .10$, where s is node 1, and t is node 18. The time to run both problems is negligible. The output from the algorithm is shown in Tables 2 and 3. (No duplicates are generated in this case.)

Arcs Crossing Optimal and Near-Optimal Cutsets	Residual 1-18 Maximum Flow
(1,7)′ (1, 2)	95
(13,14)′ (7,8)′ (1,2)	95
(7,13)′ (7, 8)′ (1,2)	95
(1,7) (1,2)′	99
(15,16) (9,10)′ (2,3)′	99
(15,16) (9,10)′ (3,4)′	99

Table 2. Output for the small network with $R = 2$ and $\varepsilon = .05$. The notation (i,j) indicates that the arc was not interdicted, and $(i,j)'$ indicates that the arc was interdicted. There are six interdictions within 10% of the maximum interdiction.

Arcs Crossing Optimal and Near-Optimal Cutsets	Residual 1-18 Maximum Flow
(1,7)′ (1, 2)	95
(13,14)′ (7,8)′ (1,2)	95
(7,13)′ (7, 8)′ (1,2)	95
(1,7) (1,2)′	99
(15,16) (9,10)′ (2,3)′	99
(15,16) (9,10)′ (3,4)′	99
(7,13)′ (7,8) (1,2)′	101
(13,14)′ (7,8) (1,2)′	101
(17,18)' (12,18)	102
(13,14) (7,8)′ (1,2)′	103
(14,15) (9,10)′ (2,3)′ duplicate	103
(14,15) (9,10)′ (3,4)′ duplicate	103
(7,13) (7,8)′ (1,2)′ duplicate	104
(17,18) (12,18)′	104

Table 3. Output for the small network with $R = 2$ and $\varepsilon = .10$.
The notation (i,j) indicates that the arc was not interdicted,
and $(i,j)'$ indicates that the arc was interdicted.

The potential usefulness of our algorithm can quickly be seen in the results shown in Table 2. There are three optimal solutions shown. However, had we not enumerated near-optimal solutions, only one of these would have been found. In addition, through partial-enumeration we identify three near-optimal solutions for $\varepsilon = .05$, and 8 near-optimal solutions for $\varepsilon = .10$. This could provide an interdictor with a wide choice of good strategies strategies to employ that also satisfy secondary or tertiary objectives.

C. COMPUTATIONAL RESULTS FOR THE LARGE NETWORKS

Here we generate three large networks and run our algorithm on each. We choose $u_{ij} \in \{40,\ldots,60\}$ and $r_{ij} \in \{1,2\}$ for each of these networks. We also choose to make all vertical arcs, and three quarters of the horizontal arcs on the outer face uninterdictable

($r_{ij} = \infty$). This allows us to run our algorithm for higher values of R without completely disconnecting s and t. For the first large network (LN1), we set $n = 10$, which creates a 10×20 network with 200 nodes and 315 arcs. The second large network (LN2) is generated by setting $n = 15$, which creates a 15×30 network with 450 nodes and 700 arcs. For the last large network we set $n = 20$, which generates a 20×40 grid network with 800 nodes and 1247 arcs. We run the algorithm for $R = 3$, 5 and 10 for all problems.

Computational results are summarized in Tables 4, 5 and 6. We do not include results for the dual-finding algorithm or Dijkstra's algorithm because those run times are negligible, less than 0.06 seconds in any case. Off-line culling of duplicate solutions is not accounted for in the run time either; there are typically very few duplicates generated. The number of near-optimal solutions excludes one that is considered "the optimal solution."

	Enumeration (secs)			**Total Time (secs)**			**Near-optimal Solutions**		
ε	R=3	R=5	R=10	R=3	R=5	R=10	R=3	R=5	R=10
.05	.00	.00	.00	.02	.02	.03	0	0	0
.10	.00	.00	.05	.02	.02	.08	0	0	18
.20	.00	.00	.28	.02	.02	.31	0	0	135
.50	.00	.00	.44	.03	.02	.47	1	1	227
1.00	.01	.01	.61	.03	.04	.64	4	9	323

Table 4. Computational Results for LN1. Solving the network interdiction problem within 20% of optimal on network LN1, with 10 units of interdiction resource, enumerates 135 near-optimal solutions in 0.31 seconds. Duplicate solutions have been eliminated.

	Enumeration (secs)			Total Time (secs)			Near-optimal Solutions		
ε	R=3	R=5	R=10	R=3	R=5	R=10	R=3	R=5	R=10
.05	.00	.00	.35	.05	.07	.42	0	5	239
.10	.00	.00	.35	.05	.07	.42	0	5	239
.20	.00	.00	.35	.05	.07	.42	0	5	239
.50	.00	.01	1.00	.05	.07	1.06	1	12	649
1.00	.02	.04	1.33	.07	.09	1.39	9	39	855

Table 5. Computational Results for LN2. . Solving the network interdiction problem within 100% of optimal on network LN2, with 5 units of interdiction resource, enumerates 39 near-optimal solutions in 0.09 seconds.

	Enumeration (secs)			Total Time (secs)			Near-optimal Solutions		
ε	R=3	R=5	R=10	R=3	R=5	R=10	R=3	R=5	R=10
.05	.01	.01	.03	.10	.10	.15	1	1	17
.10	.01	.01	.03	.10	.10	.15	1	1	17
.20	.01	.01	.03	.10	.10	.15	1	1	17
.50	.01	.01	.03	.10	.10	.15	3	3	19
1.00	.00	.01	.09	.09	.11	.20	6	10	62

Table 6. Computational Results for LN3. Solving the network interdiction problem within 50% of optimal on network LN3, with 10 units of interdiction resource, enumerates 19 near-optimal solutions in .15 seconds.

The ranges for $|N|$, $|A|$ and R are too small to result in appreciable run times for the dual-finding and Dijkstra sub-algorithms on these problems. (Of course, the dual-finding sub-algorithm is independent of R.) As expected, the run time for the enumeration sub-algorithm increases dramatically as the number of near-optimal solutions increases and as ε increases. However, that sub-algorithm is still quite efficient, even for large values of ε.

As a whole, our interdiction algorithm has proven reasonably efficient, running in less than a minute for each of our examples. It appears that the algorithm would prove an efficient tool for somewhat larger networks, and larger values of R, especially on a faster computer. The value of ε will probably be the limiting factor, and this suggests a strategy controlling the run time and sheer volume of alternate interdictions by successively increasing ε, cautiously, for larger problems.

VI. CONCLUSIONS AND RECOMMENDATIONS

This thesis has presented specialized network interdiction algorithms to solve the network interdiction problem optimally on planar networks, *and also enumerate near-optimal solutions*. The basic problem is to minimize the maximum flow in a capacitated network by destroying arcs using limited resources. We propose two distinct algorithms for the problem, the first algorithm being applicable to *s-t* planar networks (*s* and *t* must lie on the perimeter of the network), and the second to general planar networks.

The first algorithm takes as its input an adversary's primal network embedded in the plane, constructs the dual network, and then utilizes a variant of Dijkstra's shortest path algorithm on an expanded version of the dual to solve the interdiction problem optimally. The dual is expanded to deal with the consumption of varying, integral amounts of interdiction resource. Using the expanded dual network, the algorithm then enumerates all solutions within 100ε% of optimal using a modified path enumeration algorithm. The algorithm has been implemented and computational results show that the methodology is quite efficient. A significant drawback to the algorithm is that it requires the network to be *s-t* planar. We therefore propose an extension of the algorithm to handle general planar networks.

The second, more general algorithm is a fairly straightforward modification of the first except that: (a) The algorithm effectively searches for cycles in the dual network rather than paths, (b) it must keep track of the number of times (odd or even) that a cycle has crossed a specified *s-t* path, and (c) the shortest path and enumeration sub-algorithms must be called once for each arc on that *s-t* path. Though we do not implement it, the algorithm should still prove reasonably efficient as it should require (roughly) no more

than $2m$ times more work than the first algorithm, where m is the minimum number of arcs among s-t paths. A suggested modification may further reduce the factor of $2m$.

Further development of the algorithms we have proposed is warranted. The second algorithm for general planar networks should be implemented and tested. An efficient planarity testing and embedding algorithm (Hopcroft and Tarjan, 1975) should be added as a "front end" to these algorithms: Some networks may be planar and susceptible to our techniques, but no planar embedding is customarily provided. An automatic mechanism for culling duplicate candidate solutions should be appended to our code, also. We believe that a sorting procedure employing a hash function would be quite efficient and could be applied as candidate solutions are generated (and duplicate solutions never presented to the user). Modest modifications to the enumeration algorithm could eliminate some of the duplicates, too.

When a network is non-planar, our methodology is not directly applicable. However, if deleting only a few nodes would yield a planar graph, we conjecture that there is some way to extend our methods to fit this situation. The problem of finding the minimum number of nodes or arcs whose deletion yields a planar graph is NP-complete (e.g., Garey and Johnson, 1979, p.195). However, efficient methods exist for finding maximal planar subgraphs that are often close to being maximum (Cai, *et al.*, 1993). So, identification of nearly planar networks should not be an insurmountable stumbling block to extend our methods to such networks.

LIST OF REFERENCES

Ahuja, R.K., Magnati, T.L. and Orlin, J.B., Network Flows: Theory, Algorithms, and Applications, Prentice Hall, 1993.

Bhaskar, J. and Sahni, S., "A Linear Algorithm to Find a Rectangular Dual of a Planar Triangulated Graph," *Algorithmica*, **3**, No. 2, pp. 247-278, 1988.

Byers, T.H. and Waterman, M.S., "Determining All Optimal and Near-Optimal Solutions when Solving Shortest Path Problems by Dynamic Programming," *Operations Research*, **32**, No. 6, pp. 1381-1384, 1984.

Borland International, Inc., *C++ Builder 3*, 1997.

Cai, J., Han, X., Tarjan, R.E., "An $O(m \log n)$-Time Algorithm for The Maximal Planar Subgraph Problem," *SIAM Journal on Computing*, **22**, pp. 1142-1162, 1993.

Cormen, T.H., Leiserson, C.E. and Rivest, R.L., Introduction to Algorithms, MIT Press, 1992.

Cormican, K.J., "Computational Methods for Deterministic and Stochastic Network Interdiction Problems," Master's Thesis, Naval Postgraduate School, Monterey, California, March 1995.

Ford, L.R. and Fulkerson, D.R., "Flows in Networks", R-375-PR, Rand Corporation, Santa Monica, California, August 1962.

Garey, M.R., Johnson, D.S., Computers and Intractability: A Guide to the Theory of NP-Completeness, Bell Laboratories, 1979.

Hart, P.E., Nilsson, N.J., and Raphael, B. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on SSC*, SSC-4:100-107, 1968.

Hopcroft, J.E., Tarjan, R.E., "Efficient Planarity Testing," *Journal of the ACM*, **21**, pp. 549-568, 1974.

Lawler, E.L., Combinatorial Optimization: Networks and Matroids, Holt, Rinehart and Winston, 1976.

O'Rourke, J., Computational Geometry in C, Cambridge University Press, 1994.

Phillips, C.A., "The Network Destruction Problem," Working Paper, Sandia National Laboratories, Albuquerque, New Mexico, April 1992.

Preston, C.P., "Interdiction of a Transportation Network," Master's Thesis, Naval Postgraduate School, Monterey, California, March 1972.

Steinrauf, R.L., "Network Interdiction Models," Master's Thesis, Naval Postgraduate School, Monterey, California, September 1991.

Wollmer, R.D., "Removing Arcs From a Network," *Operations Research*, **12**, No. 6, pp. 934-940, December 1964.

Wood, R.K., "Deterministic Network Interdiction," *Mathematical and Computer Modeling*, **17**, No. 2, pp. 1-18, 1993.

INITIAL DISTRIBUTION LIST

No. Copies

1. Defense Technical Information Center 2
   8725 John J. Kingman Rd., Ste 0944
   Alexandria, VA 22304-6145

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, California 93943-5101

3. Professor R.K. Wood 8
   Department of Operations Research
   Naval Postgraduate School, Code OR/Wd
   Monterey, CA 93943

4. Lt. M.R. Boyle 2
   2568 Dunnigan St.
   Camarillo, CA 93010